RSTS-11 SIG
DECUS

DIGITGRAPH
P. O. Box 4249
Tucson, Arizona 85717

JULY 1977 RSTS-11 SPECIAL INTEREST GROUP NEWSLETTER VOL 4 NO 3

FROM THE SIG CHAIRMAN

What happened at Spring DECUS? Well over 500 RSTS users were treated to four full days of sessions. The main topics of discussion were how V6B is working, the need for automated patching for RSTS, measurement of system performance, and Backup under 6B.

There isn't anytime to spend recapping the sessions because the deadline for papers for Fall DECUS is upon us(August 10)!!! With over 500 RSTS users attending DECUS the SIG is faced with size problems that make it very hard to have effective sessions. The RSTS SIG steering committee has set as its major goals for the Symposium to have a wider variety of sessions that are shorter, more specialized, and more concentrated in content. To reach these goals we are appealing for your help. If you see the need for a session get a paper discription into the DECUS office, so that it will happen. We need more givers in the SIG and fewer takers.

Some of the suggested topics for sessions include: Backup techniques, system performance measurement, Basic-Plus to Basic-Plus II conversion, the experiences of users who have converted to RSTS from other brands of computers, application papers on special uses of RSTS, system security techniques, and user CUSP development.

Now that I have started my own business, I needed more help with the SIG and Murray Stinson has volunteered to serve as newsletter editor. I hope you will help Murray by sending him lots of good articles in nice camera ready form.

Bob Branton
3115 Brincrest Dr.
Dallas, Tx. 75234
(214) 243-0420

Here is a list of the hard working people who are chairing RSTS Subgroups:

A. COMMERCIAL RSTS USERS
Stu Katz
Commercial Computer Services
401 Fullerton Parkway
Chicago, IL 60614
312-929-7400

EDUCATIONAL RSTS USERS
Howard Kipple
New Mexico Junior College
Hobbs, NM 88240
505-392-6526

CUSP EDITOR
Jon Spencer
PASCAL Computing
432 S. Barrington Ave.
Los Angeles, CA 90049

JSTATS, DSTATS DOCUMENTATION
Mike Mayfield
Digital Equipment Corporation
871 South Nash St.
El Segundo, CA 90245

NEWSLETTER EDITOR
Murray Stinson
Scientific Placement, Inc.
2700 S. Post Oak Rd., #2430
Houston, TX 77056
713-621-9811

LARGE RSTS SYSTEMS GROUP
Joan Salkin
Crown Zellerbach
1 Bush Street
San Francisco, CA 94119
415-823-5504

TECHNICAL EDITOR
Carl Marbach
Acme-Hardesty Company
910 Fox Pavilion
Jenkintown, PA 19046
215-886-0200

B. WORKING GROUPS:

RSTS USER PROFILES
Bill Tabor
Florida Computer, Inc.
99 N. W. 183 St.
N. Miami, FL 33169
305-652-1710

RSTS DECUS LIBRARY PROGRAM
Ardoth Wilson
Central State University
100 N. University
Edmond, OK 73034
405-341-2980 Ext. 321

PROGRAMMING STANDARDS DEVELOPMENT
Roy Swatzell, Jr.
V. A. Hospital
Room 2 R 12
Birmingham, AL 35233
205-933-8103 Ext. 239

RSTS USER WISH LIST DEVELOPMENT
Bill Sconce
Industrial Specialties
P. O. Box 4228
Louisville, KY 40204
502-895-2910

Dear RSTS Users:

Just a note to thank you for your recent participation in the Boston DECUS Symposium.

Specifically, let me thank those of you who took the time to complete the RSTS/E Questionnaires and to submit your Wish List Items. With such a large user base, we require systematic approaches to dialoguing and understanding user problems. Your help along with that of your executive board have helped facilitate clear communication of system requirements. Let me specifically thank Bill Sconce this time for the many hours of effort that he dedicated to ensuring a well documented and categorized Wish List.

I look forward to seeing each of you in San Diego and working with the special task forces that have been set up to address specific system requirements.

Best regards,

Dian C. Pekin

Dian C. Pekin

Guilford College

5800 West Friendly Avenue, Greensboro, North Carolina 27410, 919-292-5511

May 27, 1977

Mr. Bob Branton
3115 Brincrest Drive
Dallas, Texas 75234

Dear Mr. Branton:

I have recently been appointed Director of Computer Services for Guilford College, a small coeducational institution in Greensboro, North Carolina. We are interested in establishing a general accounting system for the college.

I would appreciate it if, through the DECUS newsletter, you could put me in touch with other colleges which have worked on similar projects.

Thanks for your help and consideration.

Sincerely,

William L. McLain

William L. McLain
Director of Computer Services

WLM:pl

ross systems
incorporated

June 13, 1977

Mr. Robert A. Branton
3115 Brincrest Drive
Dallas, Texas 75234

Dear Bob:

When I talked with you in Boston, I mentioned that I would be interested in forming a timesharing SIG for the fall DECUS. I would like to pursue this area with you if possible. Basically, our firm sees quite a few companies that are setting up 'internal' timesharing departments using DEC computers to essentially replace services purchased from the outside. The objective of the proposed SIG would be to provide a common forum for these companies to exchange information and solve problems that are unique to this area. Some areas of interest are likely to be:

1. Networks - Replacement alternatives for nationwide timesharing networks.
2. Conversions - Problems and techniques in converting applications to DEC equipment.
3. Documentation - Providing documentation and training to users who have little technical expertise.
4. Support - How to support internal users who are used to a high level of support from outside vendors.

Anyone who is interested in this area and in setting up a SIG can contact me.

Very truly yours,

ROSS SYSTEMS, INC.

Kenneth Ross,
Presdient

KR:lw

1900 embarcadero • suite 108 • palo alto, california 94303 • telephone (415) 326-1054

CERVECERIA CUAUHTEMOC, S.A.

APARTADO POSTAL No. 106
TEL 75-22-00

MONTERREY, N. L. MEXICO
TELEX 03-8621

CARTA BLANCA

JUNE 14, 1977

BOB BRANTON
R LA BAR ASSOCIATE, INC.
2750 NORTHAVEN ROAD - SUITE 102
DALLAS, TEXAS 75229

Dear Mr. Branton:

Regarding Information Storage and Retrieval Systems under -- RSTS/E, we are interested on purchasing or acquiring a system offering abstracts retrievals by declarant, date, descriptor term, etc. We have a midi PDP-11/70 using BASIC and COBOL -- languages. The IS&R systems, like the New York Times, are -- commonly used in big journalisms (newspaper, magazines, etc.) companies under medium or large scale computer systems.

I will appreciate your placing a notice in the next SIG newsletter in order to set a contact with someone else interested in this kind of software systems. Also, if you know who is -- working with IS&R systems under RSTS/E, please let me Know.

Sada

LORENZO SADA
Sistemas de Intormación
Cervecería Cuauhtemoc, S.A.
Monterrey, N.L.
México

(Phone: 74-13-63)

Lawrence University

(920) 735-6681
Post Office Box 1847, Appleton, WI 54911

June 23, 1977

DECUS Program Librarian
Digital Equipment Computer
Users Society
146 Main Street
Maynard, Massachusetts, 01754

Dear Friends:

It has come to our attention that the BASIC-PLUS program (TECO6B.BAS) used to initiate TECO-11 for each user for V06B-02 of RSTS/E contains one inaccurate line of code. Please replace the dialog shown in step 3.f. of the installation instructions (page 6 of System Manager's Guide to TECO-11), which accompanied our letter to you of May 20, 1977, with the following:

```
OLD DIS:TECO6B@ <CR>
Ready
310 CHAIN R$+'.LDA$' N%
COMPILE SYØ:TECO$<4Ø> <CR>
Ready
```

The corresponding BASIC-PLUS program for V06A-02 of RSTS/E does not require (nor should it have) this alteration.

We regret this error on our part, but we feel that your disseminating this letter, reproduced as part of the DECUS write-up for TECO-11, is preferable to the delays that would be involved in our recalling and re-releasing the entire TECO-11 package.

Yours sincerely,

Michael A. Hall
Director of Computer Services

James S. Evans
Associate Professor of Chemistry

cc: Mr. Koning, Lawrence
Mr. Chernoff, DIGITAL, Maynard
Mr. Branton, RSTS SIG of DECUS
Mr. Dicks, DIGITAL, Milwaukee
Ms. Pekin, DIGITAL, Maynard

MIDDLEBURY COLLEGE
MIDDLEBURY, VERMONT
05753

DEPARTMENT OF MATHEMATICS

June 27, 1977

Mr. Larry Portner
Digital Equipment Corporation
146 Main Street
Maynard MA 01754

Dear Mr. Portner,

This letter is to provide greater detail on the matter of user access to program products which Digital in some way has had an interest, but at the current time does not wish to be involved with. There was a brief hallway discussion on this matter between yourself, Anton Chernoff of the RSTS group, and myself on the last day of the Boston Decus meeting. I would like to describe the type of programs of concern to us (the RSTS SIG); they fall into roughly three classifications.

First are programs which are distributed by DEC as part of the RSTS software license. These programs, or CUSPs, are in BASIC-PLUS source form and certain programs in this collection are modified by essentially every user of a RSTS system, e.g., the LOGON program or the system accounting programs. Some users have made extremely useful modifications to these programs and would share them with the rest of the RSTS community through DECUS, only the DEC copyright on the original programs apparently prohibits this. Since the programs in question are provided in a form which encourages user modification to tailor the programs to their own installation, it would be an enormous benefit to the user community to be able to use Decus for distributing those modifications which might have a wide range of appeal.

A second class consists of programs which DEC at some time in the past actively marketed but at present is not marketing and apparently has no intention of ever doing so. An example here is the old RSTS Sort Package or IAM Package which were once marketed as part of a Commercial Extensions Package to RSTS and now have no real use in the commercial market place, but could be beneficial in some smaller application areas.

A third class of programs consist of various "in house" programs, which could be enormously helpful to the user community, but for which DEC cannot provide the necessary funding to release as part of the standard RSTS supported software. Various text editors come to mind as examples. There is apparently some mechanism for releasing this type of program through Decus, since programs like RUNOFF, CREF, and STATS have been recently placed in the Decus Library by Martin Minow.

I hope it is clear that we are not requesting DEC to give away anything to which it has proprietary rights, but rather to make available in some way, such as through Decus, programs which even in an unsupported form could be enormously helpful to the RSTS user community. Thank you very much for your attention to this matter.

Sincerely,

James Krupp
Director of Academic
Computation

cc: Bob Branton
RSTS SIG Chairman

Sifton
PROPERTIES LIMITED
P.O. BOX 5099, TERMINAL A LONDON, ONTARIO N6A 4M8
TEL. 471-0850

July 6, 1977

Mr. Murray Stinson
Scientific Placement, Inc.
2700 South Post Oak Road
Suite 2430
Houston, TX 77056

Dear Sir:

We are a new PDP 11/70 user, anticipating delivery of our system in November, and presently engaged in developing systems remotely through another local user. If possible, I would like to place the following notice in the Newsletter:

"11/70 user operating RSTS/BASIC-PLUS would like to contact other users who have developed a Canadian Payroll system - Barry McIlwaine, Sifton Properties Limited, P. O. Box 5099 Terminal A, London, Canada N6A 4M8 (Phone 519-471-0850)".

I hope that this is possible, and thanks in advance for your help.

Yours truly,
SIFTON PROPERTIES LIMITED

J. B. McIlwaine
Manager, Data Processing

JBM:mm
c.c. Decuscope
Debug

Tucson, Arizona

Commercial Sub-Sig Newsletter

The RSTS Commercial Users meeting was attended by over 100 RSTS users whose company backgrounds included manufacturing, accounting, software houses, banking and finance, plus many other unique business related areas. Several questions were posed to the group in order to get a feel for user response in each area. (1) Any need for a small version of 6B, such as one to run on a 48k machine. -- Response indicated very little need. (2) What users plan to purchase Basic-Plus II. -- Response indicated that a majority would purchase, but most would also continue to use Basic-Plus also. (3) Any need for a subscription to a Maynard Hotline for an annual charge of about $500.00. -- Response indicated that most felt it would be useful, but too expensive.

Gary Eckroft of DEC spoke on the recurring question of the difference between RSTS/E and CTS-500. He stated that company policy on this matter was still unclear, but should be resolved soon. We all hope it is before the next DECUS.

A brief panel discussion was held concerning batch processing under RSTS. Panel members were Jim Hodges of Bloodstock Computer Services and myself.

As usual the swap table for software, etc., was a big success. The RSTS marketplace will continue in the newsletter. However, for the present, we will exclude all mention of hardware sales until that matter is resolved.

Murray Stinson
Scientific Placement, Inc.
Employment Service
2700 South Post Oak Road, #2430
Houston, TX 77056
713-621-9811

-LAST MINUTE ITEM-

Bill Tabor has developed macro code to chain from FORTRAN to BASIC-PLUS or any other program. For more info, contact Bill.

Bill Tabor
Florida Computer, Inc.
99 N. W. 183 Street
N. Miami, FL 33169
305-652-1710

DESCRIPTION	CONTACT
1.) Subroutines to phonetically code last names/for storage and retrieval via standard phonetic techniques. 2.) Screen generator programs for the Tektronix 4023 buffered CRT terminal. 3.) Complete letter writing system using paragraph/letter files to build custom upper/lower case letters from form paragraphs. System allows insertions as well as a user composed paragraph.	Murray Stinson Scientific Placement, Inc. 2700 South Post Oak, #2430 Houston, Texas 77056 713-621-9811 4/1/77
1.) DATA BOSS/2 - Data Base Management Systems and Applications Monitor. Conversational setup, an English-like Inquiry Language, fast retrieval of millions of records via up to 65 keys. DBMS provides device independence and enables I/O with any ASCII file or device.	John H. Wright Florida Computer Inc. 99 N. W. 183 Street North Miami, Florida 33169 305-652-1710 4/1/77
"FIPS" - Financial Information Processing System. An easy to use computerized accounting system designed by accountants for accountants. Includes general ledger, payroll, payables, cost accounting, financial statements, etc. Features: ● Automatic Interface to General Ledger ● Complete Audit Trail ● Unique Report Formatter ● Company Consolidation and Overhead Budget Analysis	Tim Schroeder or Mike Niernberg Environmental Developers, Inc. 13693 East Iliff Avenue Denver, Colorado 80232 303-758-4619 4/1/77
The Statistical Programs for the Social Sciences/Carleton (SPSS/C) for the PDP-11. It is written in FORTRAN IV and tested under DOS V9, RT-11, and RSTS.	Graham Kimble or Carl Henry Computer Center, Carleton College Northfield, Minnesota 55057 507-645-4431 Ext. 204, 504 4/1/77
RSTS/E OMSI SORT-1 - Fast SORT for Large Files. Machine language SORT Package compatible with DEC BASIC-Plus SORT runs up to 50 times faster than DEC version. Sorts files containing over one million records. Very attractive price with OEM and multiple CPU discounts. Runs on version 6B as normal 16k word user job without software or hardware modifications. Chain out and back or run stand-alone.	OMSI Computing 4015 S. W. Canyon Road Portland, Oregon 97221 503-248-5900 4/1/77
PDP-11 TO /370 2-WAY DATA TRANSFER - Two programs: 1.) RSTS/E V6-B Basic-Plus program to transmit/receive data files to/from IBM/370-158, using BINARY I/O, on a full-duplex line. 2.) IBM VS1 ASSEMBLER program, using BTAM, may be called by COBOL or other /370 languages, to send, receive, rename, or kill files on any PDP-11, connected by dedicated full-duplex line. This system is currently in use in our installation to transmit data files in both directions between our /370-158 and a PDP-11/34 which is 200 miles away, connected by a dedicated line, as well as to our PDP-11/70, which is in the same room, and is connected by another dedicated line. Current transmission rate is 1200 Baud.	I. F. (Bud) Dawson MacMillan Bloedel Limited 1075 West Georgia Street Vancouver, B.C. Canada V6E 3R9 7/1/77

Dear Education Users:

The DECUS meeting in Boston was very profitable for those of us in education. As one user noted there were more Education RSTS Users at the Boston DECUS meeting than RSTS Users at the Miami DECUS meeting.

The Sub Sig meeting was chaired by Mike Nadoff in the absence of Howard Kipple. Several users mentioned software packages that are being developed and the organization of the Sub Sig was discussed. There is a feeling among education users that our Sub Sig needs to be organized in such a way that although many of us cannot make both DECUS meetings we can still have a strong organization. Commendation was given to Howard Kipple on his ability to organize our efforts in Boston even though he was unable to attend. A list of RSTS schools that develop programmers was compiled to be included in the newsletter. The idea of a Sub wish list for Education Users was supported. Charles Mustain will continue to compile our Education Wish List.

It was also suggested that we announce in the newsletter that Howard Kipple has been receiving requests for a list of Education RSTS Users from commercial vendors. Howard has hesitated to release such a list without our permission. Therefore, it was suggested that anyone that wished to have their name omitted from this list should contact Howard Kipple to have their name removed.

Before one DECUS meeting is over its time to prepare for another and we encourage your participation in the San Diego meeting. Suggested topics for sessions at the DECUS meeting might be:

Education System Management

Teaching RSTS and BASIC-Plus in the classroom

A session developed around Computer involvement in fields other than business or physical science

A panel discussion among commercial and education users on preparing students for computer orientated fields

A session on simulation and gaming

MIS panel

Discussion of the problems of small systems.

John W. Nunnally
Reporting for Howard Kipple

Educational Institutions Developing Programmers

The following people at the Boston DECUS meeting ask to have their institution listed in the newsletter. There has been no set of qualifications put forth to be listed here.

Thomas W. Burtnett
Computer Center Director
Dickinson College
Carlisle, PA 17013
(717) 243-5121

James Krupp
Middlebury College
Voter Hall Computer Center
Middlebury, VT 05753

Already placed RSTS programmers with DEC Business Products and other well-known firms.

John W. Nunnally
Director of Programming
Harding College
Searcy, AR 72143
(501) 268-6161 (Ext 440)

Computer Applications major and minor offered. Students work for the school as application programmers.

Pat Scopelliti
St. Bonaventure University
Drawer Y
St. Bonaventure, NY 14778
(716) 375-2026

RSTS V4 SUBSIG

The RSTS V4 SUBSIG had its usual small but dedicated turnout at the spring session. The principle item discussed was the names and locations of users with special expertise in RSTS that might be available to V4 users to answer questions or provide assistance. I hope to publish this list subsequent to obtaining the permission of those individuals involved.

A second item of interest was the ability to extend V4 systems with 32K of memory to 30K systems thus allowing a maximum user job size of 10K. This requires a hardware modification and a software patch and assumes the users system configures to 20K or less. The modification and patch information are "second hand" through me but if you are interested I will be happy to send all the data that I have on this item.

As always, if you have anything you wish to submit to this newsletter, please call or write me:

Roy H. Swatzell, Jr.
V. A. Hospital
700 South 19th St.
Birmingham, Al. 35233
(205) 933-8101, Ext. 239

RSTS BASIC PLUS PROGRAMMING STANDARDS

The RSTS STANDARDS session had a surprisingly large turnout for a late night session and the following decisions were made:

(1) The Basic Plus standards discussed in Chicago would be distributed to the users and suggested changes would be forwarded to me. These changes would be presented to the user group for a vote and subsequent modification made to the document. This was to be done by special distribution and not included in the newsletter distribution; however, I have recently found that the standards discussed in Chicago were included in DECUS RSTS11-101 DECUS S.I.G. SEMINAR PROGRAMS and Documentation. A copy of that document was obtained and a Xerox copy of the standards chapter is attached.

(2) These standards would apply to RSTS Basic Plus code only.

(3) Submissions to Decus that conform to the standards would be so noted by the author on submission.

(4) An outline describing what goes at what line numbers would be prepared and distributed with the standards document. This will be done upon completion of the editing.

(5) A copy of the standards would be included as part of the new users software distribution and would be available through DECUS, on request, to other users as a SIG sanctional software coding convention.

If you are interested in participating in this project, please forward any changes you would like to see in the attached standards document to:

ROY H. SWATZELL, JR.
V. A. HOSPITAL, RM. 2-R-12
700 SOUTH 19TH STREET
BIRMINGHAM, AL. 35233
Phone (205) 933-8101, Ext. 239

An item on the current 'wishlist' (item #138) makes a request for "automatic CHAIN to an application program from LOGIN". This can be accomplished with a simple modification to the capability which already exists in LOGIN. By modifying the lines listed below in a similar manner, it is possible to chain to a program in the user's account if he logs-in by typing 'HELLO xxx/xxx;passwrd'. If he substitutes a comma for the slash the chaining will not be performed. Obviously there are a number of variations on this basic idea.

Ronald Paprocik
The Horsburgh & Scott Co.
5114 Hamilton Ave.
Cleveland, Ohio 44114

```
LOGIN    10:55 AM         05-Jul-77
22000    NO%=INSTR(1%,A$,";")
         \ RETURN UNLESS NO%
         \ P$=RIGHT(A$,NO%+1%)
         \ A$=LEFT(A$,NO%-1%)
         \ N1%=1% IF A$="140/140" OR A$="140/141" OR A$="140/142"
                 OR A$="160/160" OR A$="170/170"
         \ RETURN UNLESS N1%
         ! CHECK FOR ACCOUNTS WHICH CAN BE CHAINED INTO
22010    PO$="MENU"
         \ P$=P$
         \ RETURN
         ! ALL ACCOUNTS CHAIN TO PO$


Ready
```

If you would like to shorten the messages sent by 'OPSER' to the Operator's Services Console, edit the program 'OPSRUN' as indicated below. The effect should be to eliminate the date, time, and job information and print only the message on one line.

Ronald Paprocik
The Horsburgh & Scott Co.
5114 Hamilton Ave.
Cleveland, Ohio 44114

```
Ready

OPSRUN  11:44 AM        01-Jul-77
4090    P$=CHR$(10%)+CHR$(13%)+P4$+P5$
        \ P$=P$+P7$ IF (M%(I1%,7%) AND 2%) <> 0%

4100    P6$,P9$=""
        \ P8$=" "
        \ P8$=" "+P8$ IF (M%(I1%,7%) AND 2%) <> 0%
                ! 2 TABS IF REQUEST.

4110    P6$=P6$+B$(I%)
        \ P9$=P9$+P8$+P6$
        \ P$=P$+P9$
        \ P$=P$+CHR$(10%)+CHR$(13%) IF B%(I%)=0%

Ready
```

Abstract

Inter-task communication, sometimes called "send/receive" was introduced under RSTS/E with version 5B of the monitor, released in Autumn, 1973. The original facility enabled the transmission of up to 20 bytes of data from a "sending" program to a "receiving" program.

Inter-task communication remained essentially unchanged until the release of version 6B of RSTS/E, in early 1977, when the facility was significantly enhanced. Transmission of up to 532 bytes of data was allowed, and groundwork for inter-processor communication via the "DECNET" product was also included.

During this time, various users have experimented with inter-task communication, and even based significant products on the send/receive capabilities of the RSTS/E monitor. Notable among these are the DEC Spooling Package and IIS Data Base Management System. Obviously, there are many others, but I picked these as examples because I am familiar with them and because they exemplify some of the problems this paper addresses.

As more users experiment with the inter-task communication facility, and additional products are developed by both DEC and DEC users, it becomes increasingly important to develop a standard protocol to avoid the pitfalls inherent in interfacing and maintaining these products.

I have been actively and intensively working with inter-task communication since receiving version 5B of RSTS/E in 1973. During this time, I have encountered many problems with the use of the facility and have developed some simple solutions to these problems. This paper describes some of those problems and solutions.

Robert H. Jones IV
Manager of Software Development
Interactive Information Systems, Inc.
Post Office Box 37403
Cincinnati, Ohio 45222

February, 1977

CHAPTER 1

Definition of Terms

In order to describe the problems that might be encountered, I must first define some terms:

MESSAGE
PACKET
SMALL PACKET
LARGE PACKET
SENDING JOB
RECEIVING JOB
PROTOCOL
COMMAND MESSAGE
RESPONSE MESSAGE

1.1 MESSAGE

A MESSAGE is the actual data that is to be transmitted from one job to another job. The MESSAGE may be of any length from 0 to n bytes. A MESSAGE is independent of and not restricted by the way that the monitor transmits the message.

1.2 PACKET

A PACKET is the element actually transmitted by the monitor in one monitor call. A PACKET is transmitted and received by the use of SYS functions described in the RSTS/E Programmer's Guide. A PACKET con-

sists of both MESSAGE and PROTOCOL data. Depending on the MESSAGE length, multiple PACKETS may be required to transmit the MESSAGE.

The length of a PACKET is dependent on two factors: 1) whether the PACKET was transmitted using small block send/receive (RSTS/E version 6A and before) or large block send/receive (RSTS/E version 6B); And 2) the length of the MESSAGE. A PACKET transmitted using small block send/receive can contain from 0 to 20 bytes of data and will be referred to as a SMALL PACKET. A PACKET transmitted using large block send/receive can contain from 0 to 532 bytes of data and will be called a LARGE PACKET.

1.3 SENDING JOB and RECEIVING JOB

The SENDING JOB is the RSTS/E job which originates the MESSAGE. The RECEIVING JOB is the RSTS/E job which receives the message. The SENDING JOB transmits the message to the RECEIVING JOB, one PACKET at a time.

1.4 PROTOCOL

PROTOCOL is a portion of each PACKET which describes the nature of the PACKET, but not the contents of the PACKET. The PROTOCOL data describes where the PACKET is to be sent (or where it came from) and how to reconstruct the PACKET to form the completed MESSAGE. Correct PROTOCOL is required to transmit and receive the PACKET, but has nothing to do with the logical content of the MESSAGE or how the RECEIVING JOB uses the MESSAGE (*).

1.5 COMMAND MESSAGE and RESPONSE MESSAGE

(*) Most of the PROTOCOL required to send or receive a PACKET is defined in the RSTS/E Programmer's Guide. The DEC supplied PROTOCOL information is sufficient if the length of the MESSAGE never exceeds the length of a PACKET. However, most serious users of inter-task communication have found this not to be the case. Additional PROTOCOL is required and discussed in chapter three of this paper.

Definition of Terms

Although inter-task communication can be used in many different applications, virtually all MESSAGES can be classified as either COMMAND MESSAGES or RESPONSE MESSAGES. A COMMAND MESSAGE requests the RECEIVING JOB to perform some action, while a RESPONSE MESSAGE is sent only as a result of having received and processed a COMMAND MESSAGE.
The SENDING JOB may originate a COMMAND MESSSAGE at any time during the program execution. Therefore, the RECEIVING JOB may receive a COMMAND MESSAGE at any time during its execution. A COMMAND MESSAGE does not necessarily require a corresponding RESPONSE MESSAGE, but if a RESPONSE MESSAGE is required, the original SENDING JOB (the program which originated the COMMAND MESSAGE) will usually go into a sleep state until it receives the RESPONSE MESSAGE.

Correctly identifying a message as either a COMMAND MESSAGE or a RESPONSE MESSAGE and understanding the difference between the two is very important, because some of the situations which arise require that a program be able to identify whether a MESSAGE just received is a COMMAND MESSAGE from another program or a RESPONSE MESSAGE to an outstanding COMMAND MESSAGE issued by this program.

CHAPTER 2

Problems And Solutions

This chapter discusses some of the problems users have encountered when trying to implement inter-task communication and presents some solutions to those problems. The solutions presented may not be optimal for all environments, but they do work.

2.1 Sending And Receiving A Message

Problem: How do you send messages between programs?

Solution: See RSTS/E Programming Manual.

Discussion:

Messages are sent and received using SYS calls. The calls are described, with examples, in chapter 8 of the RSTS/E Programming Manual. The SYS calls described for small block send/receive (FIP 18) should not be used because they cause additional overhead.

2.2 Packet Transmission And Reconstruction

Problem: How do you transmit and receive a message that is longer than the packet length?

DIGITGRAPH
P. O. Box 4249
Tucson, Arizona 85717

Solution: Break message into multiple packets.

Discussion:

The RSTS/E monitor limits the packet length to 20 bytes (Version 6A) or 532 bytes (Version 6B) of data. Often, the message is too large to fit into one packet and, therefore, must be broken into multiple packets. Breaking the message into one or more packets, transmitting the packets to the receiving job, and correctly reconstructing the packets into the original message is called packet transmission and reconstruction.

The basic problem is quite simple and easily solved; just prefix each packet with an indicator which tells when the message is complete. This allows the receiving job to repeat the receive SYS call until the entire message has been received.

2.3 Packet Interleaving

Problem: How do you separate packets coming from several different jobs?

Solution: Buffer all packets until a complete message is received.

Discussion:

What happens if several jobs are sending multiple packet messages to the same receiving job? The packets can very easily become interleaved. For example, sending job A can transmit the first packet of a two packet message to receiving job C, then exhaust its time slice. Sending job B is the next job scheduled to run, and it sends a single packet message to job C. Job C has now received two packets: the first packet (from job A) indicated that there was at least one more packet to be transmitted, while the second packet (from job B) indicated that the message is now complete.

It is obviously a mistake to assume that job C has received a two packet message from either job A or job B. The only correct action that can be taken is to process the completed single packet message from job B and buffer the partial message from job A until that message is completed.

There may be some contention that since job A was the first to initiate contact with job C, the receiving job (job C) should not process the message from job B until it has first completed and processed the message from job A. This argument would significantly degrade the performance of job C by forcing additional scheduling by the monitor; i.e., the job would have to abdicate its time slice. And that there is no logical time dependency between jobs A and B which would necessitate such action. However, this argument should serve to reinforce the designer's awareness of the asynchronous nature of jobs operating under a timesharing environment. If a time dependency does exist between jobs A and B, this dependency can be established by direct communication between these jobs.

2.4 Buffer Contention

Problem: What do you do if there are no buffers available to transmit the message?

Solution: Retry later.

Discussion:

One of the problems that often occurs during the transmission of a multi-packet message is contention for system buffers. Although this problem is documented in the programmer's guide as an error trap on the "send" SYS function, it does deserve some note because it can easily be overlooked. There is a simple way of dealing with the problem.

As most users of RSTS/E know from experience, heavy small buffer contention can severely degrade system performance. This is especially true in an environment which uses a large volume of inter-task commun-

ication.

Whether using the large or small block send/receive facility, each packet requires at least one small buffer. A large message transmitted using small packets could completely exhaust all available small buffers, or could use up all buffers allotted for the target receiver. If either condition occurs before the message is completely transmitted, the sending job will receive an error trap 4 (message receiver is full) or 32 (no small buffers available).

The "fix" for this condition is quite simple: subsequent retries of the "send" SYS call should sleep for successively greater intervals until the SYS call completes without error. By sleeping for first 1 second, then 2 seconds, then 4 seconds, etc. the system should settle out without a significant delay, and certainly without losing any messages.

It is important that the sending job attempt to complete the transmission of the message rather than just give up and restart later, because the receiving task will already have part of the message constructed in its buffers.

There are conditions which require that the sending job perform other tasks rather than continuously retry the message send. In most cases, it is sufficient to treat that part of the message which hasn't been sent as a complete message and simply retry later.

2.5 Clearing Partial Messages

Problem: How do you prevent garbage messages or message overlap?

Solution: Clear the receiver buffer first.

Discussion:

The previous section alluded to a problem which could occur when a re-

ceiving job has an incomplete message in its buffer. If the sending job somehow aborted without completing the message it was transmitting, the partial message will remain in the buffer of the receiving job. If the next program of the same job number attempts to send a message to the same receiving job, the message will not be recreated correctly due to the partial message lingering from the previous program.

A simple method for resolving this problem is to adopt a convention that a zero length (null) packet will instruct the receiving job to clear its buffers. The effect is similar to issuing a CTRL/U to a terminal before logging in to make sure there is no garbage in the terminal's input buffer.

2.6 Intermixed Large And Small Buffers

Problem: What happens if some programs use small block messages and others use large block messages?

Solution: A common receive routine can handle this condition.

Discussion:

RSTS/E Version 6B provides support for both small and large block send/receive. A New SYS call (FIP 22) has been implemented to provide for transmission of up to 532 bytes of data in one packet. Small block send receive appears to work as it did under Version 6A and earlier, with only minor enhancements. However, the large block send/receive facility is entirely new. These enhanced capabilities provide the user with more power, but also cause some significant problems.

With 6B, it is quite possible for a program to receive a large block message from one user and a small block message from another user. There are several valid reasons why this might occur. Although small block send/receive appears to work as it did under version 6A and before, the monitor level code has been completely rewritten. All transmission actually takes place in large blocks, whether the user

actually used the large block SYS call or not. The BASIC+ Run-Time system converts all small block SYS calls into a large block call and back again. This means that other languages, such as FORTRAN, can use only large block send/receive.

It is possible for a program to receive either large block or small block messages, and even intermixed large and small block messages from the same sending job.

There is only one condition that should be met for the receiving job to correctly construct the complete message: all packets must use the protocol described in chapter 5.

2.7 Combined Terminal And Message Functions

Problem: How do you make a program receive either from a message queue or from a terminal?

Solution: Not immediately clear.

Discussion:

There are some applications which require that the program interact with both the message receiver queue and the terminal. (For purposes of this discussion, multi-drop terminals can be treated logically as a single terminal). There are two basic methods which can be used to deal with this type of problem: 1) put the program into a tight loop, successively looking at both the terminal and the receiver queue until one or the other enters a ready state; and 2) enter a "sleep" state until some activity awakens the program and then determine where the activity took place.

Both solutions have some inherent problems. The first solution involves placing the program into an almost continuous "run" state with only occasional "sleeps". The net result is that the program will exhaust greater system resources than it really needs, and it often will not process messages or terminal activity until several seconds after

the activity has occurred. Two programs which utilize this processing structure are QUEMAN and OPSER as supplied by DEC with RSTS/E Version 6B. Existing RSTS/E V06B users can attest to the efficiency with which these programs perform their tasks.

Fortunately, the other approach has fewer problems. Ideally, the monitor should provide a facility to wait for the completion of a particular event, a list of events, or any event within a list of events. RSTS/E does allow the user to enter a "sleep" state and be awakened on either terminal or message receiver activity, which partially solves the problem.

The message receive subroutine in chapter 3 will return with an error 5 and MSG.DATA$ = "" if the routine was awakened due to terminal input. The application program will have to issue a "get" or "input" verb to retrieve the line typed.

It is relevant to note that this routine did not work effectively in versions of RSTS/E prior to 6B.

2.8 Peeking For Messages

Problem: Can PEEK commands be used instead of sleeping for a message?

Solution: Yes, if you like modifying code.

Discussion:

It is possible to "peek" into the monitor's tables to determine how many messages are waiting for a particular job. The SYSTAT and DISPLY programs use peeks to gather this and other information.

In previous versions of RSTS/E, message receiver tables existed which had to be scanned to obtain information. In version V06B, this information is accessed via the JDB2 block:

```
RECEIVER% = PEEK (JDB2% + 18%).
```

The pointer to the JDB2% block for the current job is always at fixed location 528%. To find the number of messages for a specified RECEIVER%:

```
NUM.MSG% = SWAP%
    (PEEK (RECEIVER% + 14%)) AND 255%
```

2.9 DECNET/E Considerations

Problem: What happens when DECNET comes, and how do I plan for it?

Solution: Functionalize your logic and be prepared to change it.

Discussion:

Most of us have heard that the Digital Networking software is just around the corner. The initial implementation of DECNET/E (the /E is for RSTS/E) will use the same send/receive messages that local inter-task communication uses. How will DECNET/E effect existing send/receive based programs, and what changes will be necessary to interface with DECNET/E when it comes?

DECNET represents one of the most significant ventures ever undertaken by DEC. It is a major corporate effort and subject to much scrutiny - even some internal (to DEC) political bickering. As a result, what DECNET will look like when it arrives is still anybody's guess. One thing is certain, however; DECNET will not be "simple and easy to use". In fact, existing send/receive functions will be simple when compared with DECNET.

Although DECNET/E is not currently available, and there is much speculation regarding its eventual delivery dates, it is usually a good practice not to "burn your bridges behind you". Designing your systems in such a way that it will be easy to change them to make use of future facilities.

What is known at this point is that DECNET/E will require significant overhead and handshaking both at the monitor and the application program level. By designing your program so that the send/receive code is highly functionalized, you can minimize the impact of the conversion to DECNET/E. It would be a big mistake to think that the conversion effort will be trivial.

*** SUMMARY ***

This chapter has presented some of the problems that users have encountered and questions they have raised when using the inter-task communication facilities of RSTS/E. The list is by no means exhaustive, but does include all the major problems of which I am aware. As new problems arise, I will update this document and re-release it.

"The remainder of Mr. Jones' paper will be included in the next newsletter."

CHAPTER 1

BASIC-PLUS Software Conventions

1.1 INTRODUCTION

This specification defines the software conventions that BASIC-PLUS programs to be run under RSTS/E should follow. These conventions include coding and program documentation/commenting techniques. The conventions are intended to simplify software maintenance, to ease software release activities, to aid the software librarian, and to allow the use of standard function libraries.

All new BASIC-PLUS programs should follow the conventions described below. When a program is extensively modified, the new code should follow these conventions.

It should be emphasized that no claim is made that the conventions described are the "best" conventions for providing documentable, maintainable BASIC-PLUS programs. Because of the flexibility of BASIC-PLUS, a large number of different sets of conventions are equally good. However, this set of conventions reflects a combination of experience and current usage on the part of the authors and others. In the interest of compatibility, this single set of conventions is highly recommended.

The information in this document is a slightly modified version of an internal DIGITAL working document that was used to standardize code written for RSTS/E version 06B. It should be noted, however, that the standard has been used prescriptively, rather than restrictively. Overriding considerations of code efficiency and functionality prevented its application in all situations. The information in this document should not be construed as a DIGITAL Standard. Future standards for BASIC-PLUS coding may bear no resemblance to the following information.

This specification was developed for programs written for BASIC-PLUS I. Programs written for BASIC-PLUS II will require modifications to certain formats. The specific format of the line is, however, less important than the fact that a definite and intentional format is imposed on the program text.

1.2 LINE FORMATS

There are three types of line formats. The first is used for general commenting. The second type is used for comments that are used to delineate a block of code. The third type is used for all user code.

1.2.1 General Comment Lines

Any line that contains just a comment and which can be deleted from the source without affecting program execution is defined as a general comment line.

Examples of this construct are outlined in section 1.3 of this chapter.

General comment lines are in the following format:

```
NNNNN!<TAB>(TEXT)
```

Where:

NNNNN is a legal BASIC-PLUS line number.

Text is the text of the comment. This text is free form except for the special cases defined elsewhere in this document. Note that all continuation lines start with a tab.

1.2.2 Block Title Comment Lines

User code is partitioned into blocks or paragraphs (see 1.4). Each such block requires a block title comment line. User defined routines also have a block title comment line (see 1.5.1). This line has the following format:

```
NNNNN<TAB>!<LF><LF><LF><TAB>!<TAB>T I T L E<LF><LF><LF>(CR)
```

After trying many formats, this exact format was selected. Note that four spaces are needed between words for adequate separation. Note: These lines may not be removed during program compaction (see 1.2.1).

1.2.3 User Code Lines

The remaining lines in a BASIC-PLUS program are lines that contain BASIC-PLUS statements along with comments. These lines are in the following format:

```
NNNNN<TAB>(CODE LINE)          <OPTIONAL>
   <TAB>(CODE LINE)
          .
          .
          .
<TAB><TAB>!<SPACE>(COMMENT TEXT)
<TAB><TAB>!<SPACE>(COMMENT TEXT) <OPTIONAL>
          .
          .
          .
<BLANK LINE>
```

After trying many formats, this format was selected. Note that the code precedes the comments. With the advent of BASIC-PLUS II and RSTS/E V06B, the comments may precede the code. This is not recommended, however. See section 1.4 for further discussion of this format.

1.3 PROGRAM ORGANIZATION AND RESERVED LINE NUMBERS

This section describes the overall structure of a BASIC-PLUS program. Line numbers assigned to one of the blocks discussed below may be used only for that block. If, for some reason, a block is not needed in a given program, its line numbers may not be used in that program.

BASIC-PLUS supplies powerful control structures (some of them currently undocumented). Specifically, it is recommended that use be made of WHILE-NEXT and UNTIL-NEXT loops as well as FOR-NEXT loops, rather than using GOTO statements to control essentially iterative routines.

In common with general PDP-11 programming practice, "it is recommended that programs will be organized on the listing such that they flow down the page, even at the cost of an extra branch or jump."

Note that the first line of each reserved block of line numbers is in a standard format (see example). To speed coding programs, these first lines and the copyright statement are provided in an 'APPEND'able utlity file.

1.3.1 Title Block

Lines 1 to 9 are reserved for the title block. Line 1 is a block title comment line (see 1.2.2) and lines 2 through 9 are general comment lines (see 1.2.1). The textual content and format of these lines are group dependent. Section 1.7 describes one possible format for the title block.

Line 10 is reserved for the EXTEND statement, if extend mode is in effect. Where needed, lines 11 to 19 are reserved for a copyright or license statement.

1.3.2 Modification History

Lines 20-99 are to be used to maintain a modification history of the program. Line 20 is the standard block title comment line. The format of the remaining lines will be:

```
[optional line number]!<TAB>V/E<TAB>DD-MMM-YYY<TAB>PURPOSE
```

where V/E is the new version/edit number, DD-MMM-YY is the date of the modification, and purpose is a brief description of the purpose of the modification.

1.3.3 Program Description Block

1.3.3.1 General Description -

Lines 100 to 299 are reserved for a general description of the program. The description should be detailed enough so that the reader has a good functional understanding of the program. Unusual formats (such as common core fields, disk record fields, etc. that would be constantly referenced by some one debugging

this program) should be given here. The inclusion of formats that are specified in the program specification is optional; the programmer should include those formats that he feels will be most needed by maintenance personnel. (It is assumed that an external document, the "program specification", contain the most detailed information concerning formats, program design, etc.)

1.3.3.2 I/O Channels -

Lines 300 to 399 contain an ordered list of all I/O channels used by the program. The use of each channel is to be fully explained.

1.3.3.3 Variable Definitions -

Lines 400 to 799 are reserved to define every BASIC-PLUS variable used by the program. These variables are in alphabetical order. Variables which only appear in the standard routines (see section 1.2.9, 1.4.0) should not be included in this list. Also, dummy function variables used by any functions should not be included in the list (except when used elsewhere in the program.) Variables used in dimension statements are included in the list and distinguished by placing "()" immediately after the variable name (e.g., A1()). The use of the array should be fully explained either in this section or in the general description section.

1.3.3.4 Function and Subroutines Descriptions -

Lines 800 to 899 contain a list of all functions and subroutines used by the program. For each routine (user-defined or standard) used, the title and the starting line number will be included here as a general comment line. User defined routines are entered first.

1.3.4 Dimension Statements

Lines 900 to 998 are reserved for dimension statements. All dimension statements are placed here. Each array dimensioned should be briefly described in a comment line. Arrays used by included standard routines will be entered by the 'APPEND' at lines 950-998. The programmer should not use these line numbers.

1.3.5 Program Start

1.3.5.1 Error Trap -

The program starts at line 1000. The statement at this line should be:

```
1000 ON ERROR GOTO 19000
          ! SET UP STANDARD ERROR TRAP
```

1.3.5.2 Header Print -

Any program which can be invoked directly by the user through a 'RUN' command uses lines 1010-1020 to print the standard header line on the user's terminal. The format of this header line is group dependent (see Section 1.7).

1.3.6 User Code Organization

All other user code (except optionally, program completion routines) are located in the range from line number 1030 to 19999. It is recommended that lines of code be assigned line numbers in increments of 10.

1.3.6.1 Constant Definitions -

Any constants used by the program should be defined and initialized immediately after line 1030. Because of the logic of BASIC-PLUS, it is efficient to use a variable to hold often used constants, rather than using literals. If the programmer wishes to do this, the variables should be initialized here, with appropriate commenting. In some cusps (e.g., SPOOL, LOGIN) certain parameters are set as constants, and the documentation includes a description of how to change this parameter by altering the line. All such constants should be initialized in this first part of the program, and this initialization should be the only operation on the line, except the comment describing exactly what is being initialized.

All numeric constants should always be terminated with a "." or a "%", as appropriate. This eliminates a source of ambiguity in the text.

1.3.6.2 Mainline Code -

The mainline code includes all non-subroutine and non-function code. It should be paragraphed as described in section 1.4.

1.3.6.3 Program Completion--

The program completion routines (close of files, self-kill, etc.) will be a separate paragraph of code. It may reside with the main-line code (before line 10000) or it may be located at lines 32700-32760. If it resides with the main code, it is recommended that it start at line 9000. It is highly recommended that a block of line numbers be left unused after the bulk of the main-line code and before the completion code to allow for program expansion.

1.3.6.4 Programmer Defined Functions and Subroutines -

Lines 10000 to 18999 are reserved for programmer defined functions and subroutines. All such functions and subroutines should be located in this reserved space. Subroutines should come first, starting at line 10000; functions follow the subroutines. It is advised that a block of line numbers be left free between subroutines and functions to allow for future expansion.

1.3.6.5 Error Handler -

Lines 19000 to 19999 are reserved for error handling. Separate modules of th program may use their own error handlers, but the trap line number should alway be reset to 19000 upon exit. The standard routines will trap their own errors where appropriate, and reset the error trap to 19000 upon exit.

1.3.7 Standard Routines

Lines 20000 to 29999 are reserved for use by standard routines (see sectio 1.5.2).

1.3.8 Standard Entry Lines

Lines numbered 30000, 31000, and 32000 have special functions as entry-points t the program from CCL commands, 'CHAINING' to or from CUSP programs, and logge out terminals. In CUSPS, these lines are used as follows:

```
30000 CCL.ENTRY
31000 CHAIN ENTRY
32000 LOGGED OUT ENTRY
```

1.3.9 END Statement

Line 32767 is reserved for use as the "END" statement. No other line is allowe to contain an END statement, and no other statement may be present on lin 32767.

1.4 USER CODE FORMAT

This section further describes the structure of user code lines which wa introduced in section 1.2.3.

1.4.1 Line Format

The line number of a BASIC-PLUS code line will always be followed by a <TAB character. Therefore, lines of BASIC-PLUS code will be aligned at the first ta stop, except for some special cases outlined below. The following suggestion are highly recommended:

1. When a continuation line begins with a new command, put the backslas separating it from the previous command on the new line, not on th previous line. The backslash is to follow one tab character.

2. Where practical, further indent continuation lines which are part o the previous command, e.g., 'IF modifiers' which run on to a secon line.

DIGITAL
P.O. Box

3. Where practical, the code within a loop should be indented one tab from the statements controlling the loop. Indent the comment one tab further then the statements.

Except for 2 and 3 above, it is mandatory that the code be aligned one <TAB> from the left margin.

1.4.2 Comments

The primary purpose of commenting code is to lower the cost of software maintenance and release. Since this is best achieved by having all programmers comment their code in a uniform manner, the following conventions have been established. All comments should appear as complete sentences ending with a period, whenever possible.

1.4.3 Program Paragraphing

The general organization of a program is dictated by the requirements outlined in Section 1.3. Further clarification of the code should be achieved by "paragraphing", where each paragraph is a section of code that performs one cohesive task. Each such paragraph is separated from the others by means of a block title comment line (see 1.2.2).

A general description of the code within the paragraph should follow the title comment if the title comment does not sufficiently explain the task that the code performs.

1.4.4 Commenting Rules

Each line of the actual code is in the format described in section 1.4.1. Note that the following techniques are mandatory:

1. Each numbered line of code will have at least one comment line describing the effect of the code (except in the case described in 3 below). Each such numbered line shall be terminated with a blank physical line (except for 3 below). Each non-blank comment line shall have an exclamation point preceding the comment text (see example).

2. Computed GOTO and GOSUB statements should be commented such that the reason for each branch is explained.

3. Multi-statement lines are allowed only if each statement on the line contributes to a common goal. In such cases, the goal is thoroughly described in the usual comments following the paragraph of code (see 1 above). (If the techniques used in the paragraph are non-trivial, the algorithm is documented in the program specification.) Such "paragraphs" of code may be broken over several line numbers should it become necessary, without comments or extra line-feeds between the lines.

1.5 FUNCTIONS AND SUBROUTINES

Functions and subroutines when referred to as a group are called routines Lines 10000 to 18999 are reserved for user defined routines. Lines 20000 t 29999 are reserved for standard routines, and should not be used by th programmer. Although there are many reasonable ways to organize BASIC-PLU code, this separation of all routines is specified to standardize all BASIC-PLU code within DIGITAL. The routines themselves should start on line number ending in a "00" or a "50".

1.5.1 User Defined Functions and Subroutines

It is recommended that user defined subroutines be located between lines 1000 and 14999 and user defined functions between 15000 and 18999. It is suggeste that the subroutines precede the functions. Each routine has a title line Subroutines are entered by a 'GOSUB' to the title line (note that a subroutine' title line is a block title comment line (see 1.2.2)). A description shoul include data returned by the routine and any 'side effects' of the routin (e.g., non-dummy variables affected, arrays used, etc). Other information (lik variable definitions, I/O channels used, etc.) should be included in the firs 1000 lines of the program rather than in the routine. However, dummy functio parameters should be defined following the description of each function.

The first line number of the routine's code block is at least 2 greater than th last line of the descriptor block. All coding restrictions stated in th previous sections apply to code contains only the exit statement (FNEND fo functions; RETURN for subroutines); this is the only exit statement in th routine. Any routine which traps its own errors should use its first executabl line to set its error trap (note that the DEF statement is not executable an should be the only code on its line with the exception of return valu initialization). If the routine sets an error trap, the line before the exi line is always

```
ON ERROR GOTO 19000
        ! RESET ERROR TRAP.
```

Nothing else may appear on this line. If this line is present, the routine mus always exit through it.

An error handler block contains all error handling for the routine. If n trappable errors are possible in the routine, no error handler block is present The error block follows the main body of the routine (after the 'RETURN' o 'FNEND'). Note that all trapped errors must eventually exit through the 'O ERROR GOTO 19000' line.

All errors which the routine may generate must be trapped by the routine. Th error flag, E%, will be used to return information concerning the error to th calling program (see the following discussion of reserved variable names).

Programs must never, after trapping an error from inside a subroutine o function, continue processing without going through the exit line for tha routine. Code which does this can cause random errors with an extremely hig degree of debug difficulty.

Single-line functions may be used only when the operation performed will nc produce an error (under usual circumstances). An error trapped from single-line function must be considered fatal by the calling program.

1.5.2 Standard Routines

There is a high degree of similarity in the operations of different programs. For example, some programs accept and decode "command lines"; many programs do directory lookups; some access a standard data base. When similar functions are performed by two or more programs, the maintenance problems are reduced if the same routines are used in all of them. Thus, the existence of 'standard' routines.

1.5.2.1 Standard Routine Library -

The 'standard routine library' is a library of program files stored on a designated account. Each of these files holds code for a given often-used routine. The code from these files may be included in a program by 'APPEND'ING. In order to avoid line number conflicts, these routines have line numbers in the range 20000-29999, and no program may use these line numbers for any other purpose. All standard routines have unique line numbers so that the 'APPEND' operation can be used.

1.5.2.2 Commenting Standard Routines -

Standard routines will have the same documentation structure as does a program and will follow the same rules as user defined routines (see 1.5.1). Specifically, each routine will have a title block, a modification history, and a program description block. All standard routines which use an (internal) array will include the DIM statement for that array.

Remember that lines 951-998 are reserved for this purpose. If the copyright statement is desired, it shall use lines 11 to 19. The programmer is free to use line numbers of his choice for these items except the first line of the title block should fall on a line number that is a multiple of 50. See section 1.6 for rules on variables permitted in standard routines.

1.6 RESERVED VARIABLE NAMES

Because of the lack of 'local' variables in BASIC-PLUS, any non-dummy variables used in a function or subroutine can affect the whole program. Because not all variables in standard routines can be dummy variables, some variable names are reserved to the standard routines and should not be indiscriminately used in the user part of the code, except as dictated by the standard routines. In addition to these variables, certain other variables have been reserved to hold values which will be needed in (almost) every program. These variables must never be used to hold anything but the contents described below.

All variables of all types (integer, floating, string, or arrays of these) whose names start with 'Y' or 'Z' are reserved for use by standard routines.

All 'X' variables of all types are reserved for use as debugging variables. No other use may be made of these variables. They may only be used in explicitly debugging-type statments in the code. X variables will appear in released programs only in such statements.

The two variables 'E%' and 'E$' are used only to hold information concerning the

latest error encountered. If, at any time during a program run, E% is non-zero it indicates that an error condition exists. When E% is used and is 0, no erro condition has yet been detected. The standard routines all conform to thi rule, and all user routines must conform as well. The string in E$ can be an error-related text. If additional error information is required, the variabl E1% may be used to store it. E1%, however, may be used for other things by th program. An error processing routine must reset E% before returning control t normal processing.

Other variables may be reserved on a group dependent basis.

1.7 RSTS/E VERSION 06B DEPENDENT STANDARDS

This section describes programming conventions followed when coding System Programs for RSTS/E Version 06B.

1.7.1 Title Block

Line 1 of every BASIC-PLUS program is a block title comment line (see sectio 1.2.2). The text of this line is the name of the program. Lines 2 through 1 are general comment lines (see section 1.2.1) as shown below:

```
2!<TAB><TAB>PROGRAM          :   PROGNM
3!<TAB><TAB>PACKAGE          :   PACKNM/VEN
4!<TAB><TAB>MODULE           :   N OF M
5!<TAB><TAB>VERSION          :   VER
6!<TAB><TAB>EDIT             :   ED
7!<TAB><TAB>EDIT DATE        :   DD/MM/YY
8!
9!<TAB><TAB>AUTHOR           :   NAME<LF><CR>
```

Where: PROGNM is the program name, PACKNM/VEN is the name and version numbe of the package to which this program belongs; N of M is the sequenc number (N) of this program within the package and the number o programs (M) in the package; VER and ED are the version and edi. number of this program (see below); and NAME is the name or initial of the programmer.

If the program is not part of a package, lines 3 and 4 do not appear Line 9 does not appear in the release copy of the program (it i removed during the kit generation process - the in-house maste sources will always have it).

If the program uses the extend-mode available with BASIC-PLUS under RSTS/E V06B line 10 is present with the format:

```
10<TAB>EXTEND<LF><CR>
```

The format of the version identifier of a program is determined as outlined i the PDP-11 software engineering software standards document. It is th programmer's responsibility to make certain that the key-letter is changed t 'V' before final packaging. Since an 'EDIT' and a 'PATCH' are identical i BASIC-PLUS, the 'PATCH' field is not used.

The version number of a new program is always identical to the version number o

RSTS/E under which it will first be released. The version number of an old program may change in either of the following two cases:

1. The program is re-written from scratch, and uses capabilites of later releases of RSTS/E.

2. The program is upgraded to use features available in a later release of RSTS/E.

In any other modification, for example, bug fixes or extensions which will work under the original RSTS/E, only the edit level will change. Thus, it is possible to determine, from the version number of the program, the earliest version of the operating system under which the program can successfully execute.

Version numbers are assigned to packages in the same manner as they are assigned to programs. Note that if the version number of any program in a package increases, the version number of the package must also increase, since the package will no longer run under the earlier version of RSTS/E.

1.7.2 Title Printing

Any program which can be invoked directly by the user through a 'RUN' command uses lines 1010-1020 to print the standard header line on the user's terminal. The header line shall show the following information:

```
PRGNAM<TAB>V/E<TAB>SYSVEN
```

where PRGNAM is the name and V/E is the version/edit of the program being run, and SYSVEN is the system identification obtained from the 0th error slot in 'ERR.SYS'. At line 1010, the variable I$ will be set equal to the text defining the V/E number of the program; e.g.,

```
1010 I$="V06A-01"
          ! SET UP VERSION/EDIT FOR HEADER.
```

Line 1020 will do the final creation and printing of the header. If the program may not be invoked directly by a 'RUN' command, lines 1010-1020 should not be used. Separating the V/E out onto a separate line makes it easier to update this string upon patching the program.

Ordinarily, a program which has been entered through a 'CHAIN' or a CCL command will not print its header line. The entry line combined with line 1020 should contain the code to inhibit the printing in this case.

1.7.3 Additional Reserved Variable Names

The value of variable E0% will indicate the type of entry (i.e., RUN, CCL, CHAIN, or LOGGED-OUT entry) made to the program. The primary uses of this variable are:

1. To determine whether or not to print the header line.

2. To determine how to prompt and what editing features to allow in the standard GET command line routine.

Since the "get command line" routine has not been finalized, all values for E0% are not yet known; however, an E0% value of 0 always indicates a run-type entry. In this case, the header line should be printed and GCML should prompt fully.

1.7.4 Forbidden Usage

The following 4 features of BASIC-PLUS may not be used when writing general-usage programs:

1. 'E format' numeric constants. All numeric constants are written out in full; e.g.,

   ```
   1.E10 should not be used.  Use
   100000000000.  instead.
   ```

2. String-arithmetic and extended mathematical functions.

3. 'PRINT USING' and 'MAT' statements.

4. Long message SEND/RECEIVE.

5. Echo control on KB: handling.

In short, no "optional features" of BASIC-PLUS may be assumed in any CUSP.

1.7.5 Implied/Default Values

In general, no "defaults" supplied by BASIC-PLUS should be used. This rule is reflected in the requirement that numeric constant are terminated by "." or "%". The following further restrictions also apply:

1. Semicolons in PRINT statements will always be explicitly entered; e.g.,

   ```
   PRINT A "TEST" B$
   ```

   Is incorrect. Use

   ```
   PRINT A;  "TEST";  B$
   ```

   instead.

2. Never use "&" to mean PRINT (BASIC-PLUS II uses & for various things, but not as a synonym for PRINT).

3. All OPEN statements contain a filename and extension specification, even if they refer to a non-file-structured device.

   E.g.,

   ```
   OPEN "KB:" AS ...  is illegal, but

   OPEN "KB:PIP.CMD"...  is legal.
   ```

4. Spaces are used in the usual places; space removal to compact lines is not allowed under any circumstances. (BASIC-PLUS II will require spaces.)

1.7.6 Miscellaneous Requirements

1. In order to provide easier transition to BASIC-PLUS II, colon (:) will no longer be used as a statement separator. Instead, the backslash (\) character will be used.

2. Upon CCL, logged-out, or RUN entry, each program should execute the following two statements:

```
ON ERROR GOTO 19000
\ PRINT IF POS(0%)<>0%
```

   Setting the error trap and returning the KB: to the left margin before any other processing takes place. (Note, in the RUN entry, the PRINT IF POS(0%)<>0% command occurs on the same line as the header print.)

3. Because of the new CCL mechanism, all programs which have a CCL entry must check for a full length CCL command to extract the characters of the command.

4. BASIC-PLUS uses the "!" character only as a beginning of comment delimiter. BASIC-PLUS II, however, will use the "!" character as both a beginning and end of comment delimiter (i.e. BASIC-PLUS II will allow imbedded comments). Hence, only one exclamation point may appear in any BASIC-PLUS comment line.